1. **What are the addresses of the following register in data space and IO space (i)EECR, (ii)SREG, (iii)PINA**

Register	Address in data space XXH	Address in IO space XXH
EECR	*3C*	*1C*
SREG	*5F*	*3F*
PINA	*39*	*19*

2. **Use Atmega16 Instructions in both IO mode and memory direct mode to transfer data between R0 and these register as shown in the following tble**

	Operation	IO mode instruction	memory-direct mode instruction
i	**R0 ←PINA**	*IN R0, PINA*	*LDS R0, 0x39*
ii	**TCNT0← R0**	*OUT TCNT0, R0*	*STS 0x52, R0*
iii	**R0 ← R5**	*cannot be done in IO*	*STS 0x05, R0*

3. **What are the values needed for CKSEL bits (CKSEL3, CKSEL2, CKSEL1, CKSEL0) for the following situations**

Situation	CKSEL3, CKSEL2, CKSEL1, CKSEL0
(i) 5MHz external RC oscillator	*1110*
(ii) 8MHz external clock	*0000*
(iii) 4MHz internal RC oscillator	*0011*
(iv) 500 KHz external ceramic osc	*1011*
(v) 2MHz external crystal oscillator	*1101*

4. **What are the maximum delays could be made by ATmega16 timer0 when CPU works with the highest clock frequency in: (i) normal mode? (ii) CTC mode?**

(i) In normal mode, max delay is the time needed to count 256 times with the lowest counting rate
For lowest clock, N= 1024 (FCPU divided by maximum prescaler)
FCPU highest clock = 16MHz
*Max delay = 256*N/FCPU = (256*1024)/(16*10^6) = 16.38 msec*
(ii) In CTC mode, max delay is the time needed to count with the lowest counting rate until reach OCR. OCR is to be loaded with the largest value (255).
For lowest clock, N= 1024 (FCPU divided by maximum prescaler)
FCPU highest clock = 16MHz
*Max delay = (1+OCR)*N/FCPU = (255+1)*1024/(16*10^6) = 16.38 msec*

5. **What is the delay that is made by ATmega16 timer0 for each of the following conditions?**

(i) TCCR0 = 03H, OCR0 = 7FH, FCPU = 8MHz
30H in TCCR0 => normal mode, N = 64
*Delay is the time needed to count until overflow = 256*N/FCPU = 256*64/8M = 2.048 msec*
(ii) TCCR0 = 0AH, OCR0 = 7FH, FCPU = 4MHz
0AH in TCCR0 => CTC mode, N = 8
*Delay is the time needed to count until match OCR (127) = (1+OCR)*N/FCPU = 128*8/4M = 0.256 msec*

6. **What is the binary configuration needed for TCCR and OCR0 to make a 16mses delay when FCPU is 1MHz in (i) normal mode (ii) CTC mode**

(i) We need to find the value of N (prescaler) where:
*16m = 256*N/1M => N = 16*10^{-3}*10^6/256 = 62.5 ≈ 64*
TCCR binary configuration: 0 0 0 0 0 0 1 1 (normal mode, prescaler is 64)
OCR vlue: X X X X X X X X (not needed in normal mode)

(ii) We need to find a value of N (prescaler) and a value of OCR where:
*16m = (1+OCR)*N/1M =>*
In this case we often assume tries a value of N and calculate the corresponding OCR value. If the calculated value is in the range 0-255 then that is the solution, otherwise we tries another value of N until a solution if found. You can guess from the previous problem that N ≥ 64
Lets choose N = 64
*OCR = -1 + (16*10^{-3}*10^{6})/64 = 249*
TCCR binary configuration: 0 0 0 0 1 0 1 1 (CTC mode, prescaler is 64)
OCR vlue: 1 1 1 1 0 1 1 1 (OCR = 249)

7. Sketch the square wave generated out of OC0 when OCR0 = 191, TCCR0 = 7BH, FCPU = 1MHz.

TCCR0 = 7BH => fast PWM mode, set OC0 in compare match (inverse mode), prescaler = 64
*T = 256*N/FCPU = 256*64/1M = 16.38 msec*
*T_{OFF} = (1+OCR)*N/FCPU = 192*64/1M = 12.29msec*
T_{ON} = T - T_{OFF}= 16.38 – 12.29 = 4.09msec

12.29 msec 4.09 msec

8. Name the flag and its register that is used to indicate the following statuses in ATmega16

	Event or command	bit name	register
i	**timer2 counter rolls from 255 to zero**	*TOV2*	*TIFR*
ii	**timer0 counter reaches OCR0**	*OCF0*	*TIFR*
iii	**completion of ADC conversion**	*SC or ADIF*	*ADCSRA*
iv	**completion of EEPROM write**	*EEWE*	*EECR*
v	**overflow of the last ALU result**	*V*	*SREG*

9. Name the bit and its register that is used to initiate the following actions in ATmega16

	Event or command	bit name	register
i	**EEPROM read**	*EERE*	*EECR*
ii	**ADC start conversion**	*SC*	*ADCSRA*
iii	**enable interrupts**	*I*	*SREG*
iv	**enable interrupt when ADC completes the current conversion**	*ADEN*	*ADCSRA*
v	**enable interrupt when timer0 counter rolls from 255 to 0**	*TOIE0*	*TIMSK*

10. Write ATmega16 instructions to do the following jobs

1- copy contents of R1 to R6

```
MOV   R6, R1
```

2- Put zero in R10

```
CLR   R10
```

3- Add 24 to R5

```
LDI   R16, 24
ADD   R5, R16
```

4- Add 50 to R20

```
ADI   R20, 50
```

5- Get into R0 a byte from port C

```
IN    R0, PINC
```

6- Output 55H on port B.

```
LDI   R19, 0x55
OUT   PORTB, R19
```

7- Store contents of R8 into location 150H of SRAM.

```
STS   0x150, R8
```

8- Load R6 with a byte from memory location 060H.

```
LDS   R6, 0x060
```

9- Increment memory location 100H

```
LDS   R0, 0x100
INC   R0
STS   0x100, R0
```

10- Store "70H" into SRAM memory location where R1:R0 points

```
LDI   R0, 0x70
MOVW  R1:R0, R27:R26
ST    X, R0
```

11- Load R17 with contents of memory location pointed by Y

```
LD    R17, Y
```

12- Load R0 with a byte from memory location pointed by Y+15

```
LDD   R0, Y+15
```

13- Store bit3 of R10 into T bit of SREG

```
BST   R10, 3
```

14- Configure port A upper bits (A7-A4) as inputs with pull-up resistors and lower bits as outputs initially high.

```
LDI   R16, 0x0F
OUT   DDRA, R16
SER   R20
OUT   PORTA, R20
```

15- Clear R18 if it is greater than or equal 20

```
CPI   R18, 20
BRL   NEXT
CLR   R18
NEXT:
```

16- Load program counter with 400H

```
JMP   400H
```

17- Add 60 to program counter

```
RJMP  60
```

18- Go absolutely to address "THERE" if bit5 of R5 is set

```
LDI   R16, 0b00100000
AND   R16, R5
BREQ  THERE
```

19- Set bit3 of R7

```
MOV   R16, R7
ORI   R16, 0b0000 1000
MOV   R7, R16
```

20- Clear ADE bit in ADCSRA

```
CBIO  ADCSRA, 7
```

21- Set SC bit in ADCSRA

```
SBI   ADCSRA, 6
```

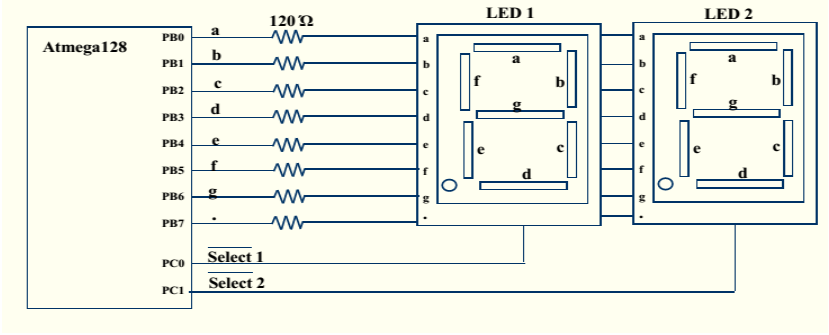
22- Continuously check SC bit of ADCSRA until it is reset

```
WAIT: SBIS  ADCSRA, 6
      RJMP  WAIT
```

23- Give equivalent instruction(s) to "PUSH R1"

```
IN    R26, SPL
IN    R27, SPH
SBIW  R27:R26, 1
OUT   SPL, R26
OUT   SPH, R27
ST    X, R1
```

11. Write an assembly language program to read the 7-seg codes stored in SRAM in addresses Digit1 and Digit2 and display them in LED1 and LED2 as shown in the figure below.

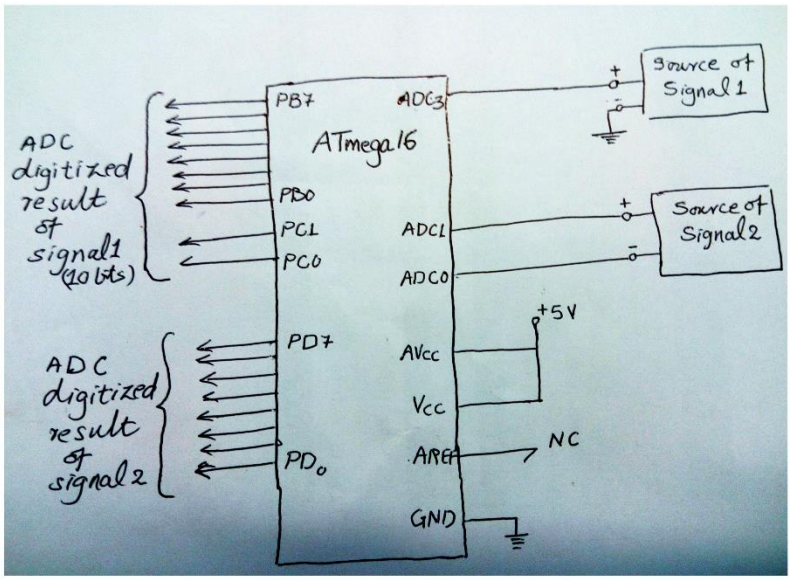



```
        SER     R0          ; control word FFH to set all bits of a port as output
        OUT     DDRB, R0    ; configure port B as output port
        OUT     DDRC, R0    ; configure port C as output port
        LDS     R1, DIGIT1  ; load digit1 into R1
        LDI     R11, 0xFE   ; load into R11 a value that would select LED1 when it is output to port C
        LDS     R2, DIGIT2  ; load digit2 into R2
        LDI     R22, 0xF    ; load into R22 a value that would select LED2 when it is output to port C
AGAIN:  OUT     PORTC, R11  ; select LED1
        OUT     PORTB, R1   ; out digit1 on LED1
        OUT     PORTC, R22  ; select LED2
        OUT     PORTB, R2   ; out digit2 on LED2
        RJMP    AGAIN       ; repeat forever
```

12. We want to use ATmega16 that runs with 8MHz to digitize two audio analog signals with a highest possible sampling rate and maximum resolution and output the digitized values on its ports. The first signal is single input within the range 0-4V. The second signal is a deferential input within the range 1 – 10 mV.

(i) Propose a suitable interface circuit.

(ii) Write a simple commented code to do the job.



- *We will choose a control word and write it in ADCSRA in order to configure ADC as follows:*

bit#	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
bit name	*ADEN*	*ADSC*	*ADATE*	*ADIF*	*ADIE*	*ADPS2*	*ADPS1*	*ADPS0*
	1	*0*	*0*	*X*	*0*	*0*	*0*	*0*
	enable ADC	*command of start conversion comes later*	*single conversion*		*disable interrupt*	*the prescaler that gives highest speed*		

ADCSRA = 80H

The first signal has a signal range (0-4V) then the optimum voltage reference source is the chip VCC. Hence, for the first signal, the control word to be written in ADMUX register would be as follows:

bit#	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
bit name	*REFS1*	*REFS2*	*ADLAR*	*MUX4*	*MUX3*	*MUX2*	*MUX1*	*MUX0*
	0	*1*	*0*	*0*	*0*	*0*	*1*	*1*
	AVCC reference		*default: right adjusted*		*let's choose single ended channel 3: ADC3*			

$ADMUX_1$ = 43H

The second signal has a signal range (1-10mV) and it is differential. Clearly this range is too small for any available voltage reference. However, if it is amplified by 200 then the range becomes (0.2 – 2V) and then the optimum voltage reference source is the internal 2.56 reference voltage. Since the resolution of a 200 gain differential gain is 7bits then it is better for the ADC result to be left adjusted and only ADCH is to be read. Hence, for the second signal, the control word to be written in ADMUX register would be as follows:

bit#	*7*	*6*	*5*	*4*	*3*	*2*	*1*	*0*
bit name	*REFS1*	*REFS2*	*ADLAR*	*MUX4*	*MUX3*	*MUX2*	*MUX1*	*MUX0*
	1	*1*	*1*	*0*	*1*	*0*	*1*	*1*
	2.56 internal voltage reference		*result is better left adjusted*	*let's choose differential input with 200 gain: +ADC1 _ -ADC0*				

$ADMUX_2$ = EBH

```
; initialization
        CLR     R0          ; R0 = 0
        SER     R1          ; R1 = FFH
        OUT     DDRA, R0    ; Set port A as input
        OUT     DDRB, R1    ; Set port B as output
        OUT     DDRC, R1    ; Set port C as output
        OUT     DDRD, R11   ; Set port D as output
        LDI     R3, 0x80    ; write a control word into ADCSRA to: enable ADC, single conversion, disable
        OUT     ADCSRA, R3  ;interrupt, set ADC clock as FCPU/2
        LDI     R11, 0x43   ; R11= 43H, control word to update ADMUX to make a conversion of signal1:
                            ; AVCC reference ; result right adjusted, single ended input from ADC3
        LDI     R22, 0x43   ; R22= EBH, control word to update ADMUX to make a conversion of signal2:
                            ; internal reference ; result left adjusted, differential input +ADC1, -ADC0 with
                            ; 200 gain
```

```
; conversion for the first signal
START:  OUT   ADMUX, R11    ; update ADMUX for a conversion for signal1
        SBI   ADCSRA, 6     ; start conversion by setting SC bit in ADCSRA
WAIT1:  SBIC  ADCSRA, 6     ; Wait until conversion complete
        RJMP  WAIT1
        IN    R4, ADCL      ; Read ADC Result (ADCL then ADCH)
        IN    R5, ADCH
        OUT   PORTB, R4     ; Out result on ports B and C
        OUT   PORTC, R5
; conversion for the second signal
        OUT   ADMUX, R22    ; update ADMUX for a conversion for signal2
        SBI   ADCSRA, 6     ; start conversion by setting SC bit in ADCSRA
WAIT2:  SBIC  ADCSRA, 6     ; Wait until conversion complete
        RJMP  WAIT2
        IN    R4, ADCH      ; Read ADC Result (only ADCH)
        OUT   PORTD, R4     ; Out result on ports D
        RJMP  START
```